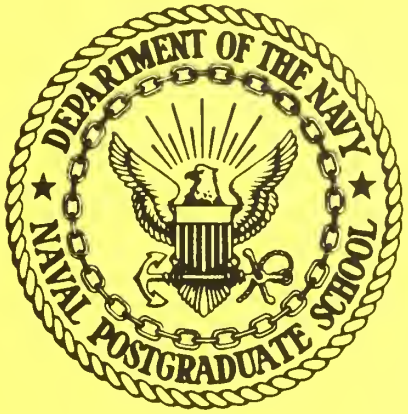
NAVAL POSTGRADUATE SCHOOL

Monterey, California



THE SPECIFICATION OF ALGORITHMS

Richard Franke

July 1975

Technical Report For Period

April 1975-June 1975

Approved for public release; distribution unlimited

ed for:
ed Digital Systems Architecture Division
Electronics Laboratory Center
ego, California 92152

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder
Superintendent

Jack R. Borsting
Provost

The work reported herein was supported in part by the Advanced Digital Systems Architecture Division of the Naval Electronics Laboratory Center.

This report was prepared by:

L.D. KOVACH
Chairman, Mathematics Department

ROBERT R. FOSSUM
Dean of Research

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS-53Fe75071	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): THE SPECIFICATION OF ALGORITHMS	5. TYPE OF REPORT & PERIOD COVERED: Technical Report, April 1975 - June 1975
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): RICHARD FRANKE	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93941	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: N0095375WR00117
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Digital Systems Architecture Division, Naval Electronics Laboratory Center, San Diego, California 92152	12. REPORT DATE: July 1975 13. NUMBER OF PAGES: 47
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

algorithm, specification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Methods for the specification of algorithms are explored. A discussion of desirable features is given and a method based on conventional ideas is proposed. Examples are given, followed by a discussion of shortcomings and related problems.

TABLE OF CONTENTS

1.0 Introduction

The primary purpose of this study was to develop a standard format for the specification of algorithms. While a number of standards exist for so-called algorithm specification, (e.g. ACM's Algorithm Policy, CACM 14(1971) 676), in fact those standards are for programs, algorithms which have been implemented in a particular language or for a particular computer.

It was desired to have a standard specification from which the algorithm can be easily implemented and from which a prospective user could decide within certain limits, whether a given algorithm meets his requirements before actual implementation on his hardware. Alternatively the algorithm specification should be able to help the potential user decide upon what hardware would be required to solve his problem. It is clear, then, that we wish to keep the specification of the algorithm independent of any particular programming language or any computer. While this is desirable from the standpoint of the user who is in the process of choosing hardware, it is more difficult for the algorithm specification to be written, and some information which it is desirable to know cannot be given in this generality.

A number of investigations into methods of specifying algorithms have been made, and we will mention some of them briefly.

D. L. Parnas [11] has suggested an algol-like language which is designed to convey to the user and the implementer alike exactly that information he needs and no more. In addition, the specification is designed so that it is possible to machine test it for completeness and consistency. The author notes that the technique has been met with some resistance and initial failures, although it "is eventually mastered by almost everyone." This writer certainly believes the first part of that statement and feels that the information conveyed about what function is performed by the algorithm is too sparse.

In the examples it is quite unclear what is being done, although perhaps this is in line with the stated intent of the author concerning the information to be conveyed. The goal of minimal information about the structure of connecting modules, and the possibility of machine testing for completeness and consistency are certainly desirable and further development may yield better methods.

The specification of algorithms in terms of a set language has been treated by several authors, e.g. [4], [14]. While these abstract languages have desirable properties, particularly in terms of precision, as Schwartz [14] notes they do not give as clear an explanation of the algorithm as a natural language description. This occurs in an elusive, but still real, sense.

The use of flow charts [16] and D-charts [1] have been discussed also. While flow charts are familiar to nearly everyone, and can be used to completely specify the flow of control in the algorithm, some important information is suppressed by them. In particular, a flow chart can show only one of perhaps many equivalent implementations of an algorithm. Possible parallelism and much of the data dependencies are obscured.

Several attempts have been made to develop methods which overcome the problems inherent in flow charts. Those methods take the form of directed graphs and are generally descendents of Petri nets [5], [15]. The most elegant of these appears to be the data flow language of Dennis and Fosseen [2], although a number of earlier versions are included in the bibliography. The data flow language has the desirable property of showing the flow of the data and is particularly useful in blocks where there are no iteration loops and no branching is required. A partial ordering on the sequence of operations is automatically induced, and possible parallelism is exposed. In addition, data dependencies are

clearly shown which is helpful in the partitioning problem, although that is not our principal interest. The inclusion of control links in the representation, necessary to show iteration loops and branching, clouds the picture considerably. For all but the simplest algorithms, the representation becomes quite clumsy, making it fairly difficult to specify the algorithm and more difficult to understand it. At a point where it would conceivably be possible for the language to be compiled by a computer, it could very well become quite valuable. Studies are being made toward a computer which could implement algorithms in this way [3].

It is recognized that it is desirable to specify algorithms in a non-redundant manner, and in a way which is capable of being machine processed for completeness and consistency. At this time it is felt that there is no method which meets all desirable requirements. Because of the distinct sets of persons who would be looking at the specification, in particular the prospective user and the implementer, it is almost a necessity that the information be somewhat redundant or that the interests of one of the two groups be treated more lightly than the other. Additionally while flow graphs or flow charts serve to specify the algorithm for the implementer's purposes, as do the mathematical equations or natural language descriptions, the sets of information in those two items serve to complement each other to a greater degree than their redundancy detracts from them. One reason for avoiding redundancy is that it allows the possibility of contradictions, which must not be allowed to occur without a method of resolving them. Even then it is not a desirable situation.

Because of the shortcomings of most new methods proposed for algorithm specification, we have attempted to maintain a somewhat conventional method of algorithm specification, while incorporating the best features of some new ideas. Further development in any of several possible directions may lead to new methods more suited to our purposes and we must certainly be alert to any such development.

2.0 The Format of the Proposed Specification

The specification of the algorithm will be in two parts, for our purposes. Those items in Section 1 will be mainly of interest to the user. Section 2 will be devoted to information needed by the implementer. After implementation, the program documentation will give information concerning storage requirements and timing which will be of interest to the user. In effect, this documentation will form a Section 3. There will be a different documentation for each implementation (different language or different computer). We will not concern ourselves with the exact form of the documentation, but it will consist primarily of the exact method of communication with the calling program, and memory and timing requirements. The implementation must conform to the specification in Sections 1 and 2 with no exceptions or else it cannot be considered to be an implementation of that algorithm.

The description of the various sections in the algorithm specification and the information they are to convey follows in Section 2.1.

2.1 The Algorithm Specification

1.0 Background Information and General Description

This section will contain information about where the algorithm is used and a brief description of what the algorithm accomplishes. The information given should be sufficient to allow the user to determine whether the algorithm is of interest to him without giving unnecessary details.

1.1 Other Algorithms Used

This section will contain a listing of other algorithms used by the subject algorithm.

1.2 Input Values

This section will contain a description of the values which are input to the algorithm. These must be described in terms of units

(where required; sometimes the input units needn't be specified and output units are the same as input units), reference lines or angles, which direction is positive, and so on. Any assumptions concerning the range in which the inputs must lie should be delineated. This, in conjunction with possible error indications (see Section 1.3) will convey to the user the possible need for preprocessing of arguments by the calling program.

1.3 Output Values and Accuracy

The values which are output from the algorithm are described in this section. Information about the accuracy of the output values should assume the inputs are exact, even though in the usual instance this will not be true. Given in this fashion, though, and given information about the accuracy of the input values, one can determine limits on the error propogated through the algorithm. When necessary, the units, reference lines or angles, positive directions, and other data must be specified as they are for input. Possible output values include error indications and potential error flags should be fully described.

1.4 Operation Count

The precise timing for an algorithm is available only after the choice of implementation has been made. However, some useful information can be given in terms of the number of operations of different type which occur and the number of references to external routines (other algorithms). Thus it will be possible to give the user a rough estimate of the execution time in terms of the operations required. This estimate will be most accurate when large numbers of arithmetic instructions are required for which the time can be estimated with some accuracy. In other instances the count may be highly dependent on the instruction repertoire of the computer, and

2.0 The Format of the Proposed Specification

The specification of the algorithm will be in two parts, for our purposes. Those items in Section 1 will be mainly of interest to the user. Section 2 will be devoted to information needed by the implementer. After implementation, the program documentation will give information concerning storage requirements and timing which will be of interest to the user. In effect, this documentation will form a Section 3. There will be a different documentation for each implementation (different language or different computer). We will not concern ourselves with the exact form of the documentation, but it will consist primarily of the exact method of communication with the calling program, and memory and timing requirements. The implementation must conform to the specification in Sections 1 and 2 with no exceptions or else it cannot be considered to be an implementation of that algorithm.

The description of the various sections in the algorithm specification and the information they are to convey follows in Section 2.1.

2.1 The Algorithm Specification

1.0 Background Information and General Description

This section will contain information about where the algorithm is used and a brief description of what the algorithm accomplishes. The information given should be sufficient to allow the user to determine whether the algorithm is of interest to him without giving unnecessary details.

1.1 Other Algorithms Used

This section will contain a listing of other algorithms used by the subject algorithm.

1.2 Input Values

This section will contain a description of the values which are input to the algorithm. These must be described in terms of units

hence only a poor estimate can be given before implementation. In any case the estimate will not include overhead and should be interpreted in that light.

1.5 Memory Requirements

This section, like the previous one, cannot be made very precise. In particular, the program length cannot be estimated without knowing the instruction repertoire of the computer on which the algorithm is implemented. However, this section can contain information concerning the number of constants required and the number of temporary storage locations required by the algorithm.

2.0 Mathematical and Natural Language Description

This section gives a full description of the algorithm in terms of the mathematic equations, if any, and the necessary natural language descriptions necessary to make clear the functions performed by the algorithm. The definition of each variable should be given. For algorithms with many variables a list should be given as this would facilitate finding the definitions easily. If possible, the range of values which may be taken on by each variable should be specified. One must at least specify whether the values taken on are integers, real numbers, complex numbers, or logical values.

2.1 Flow Diagrams

The flow diagrams may consist of two types. When the algorithm involves iterations or a large number of decisions, a conventional flow chart is probably best since in this case the flow of control is most important. When only a few decisions are required, or none, the use of a process graph is desirable since it exposes many features of which the implementer should be aware.

The type of process graph we generally have in mind is similar

to the data flow graph of Dennis and Fosseen [2] when there are no decisions to be shown. We will briefly discuss the elements and structure of the process graph as we intend it to be used.

The process graph consists of computational blocks, each of which may be as complex (a subalgorithm) or as simple (a single operation) as one likes or finds necessary. These blocks are connected by directed line segments to form a directed graph. The line segments show the flow of data through the graph. Input values to a computational block are shown as labeled line segments directed toward the block. The computations in a block may be performed as soon as all input values are available. Outputs from a block are shown by labeled line segments directed away from the block. Input values which are not output values from another block are inputs to the algorithm, except for constants which are shown arising from disks. Output values which are not input values for another block are outputs from the algorithm. Process graphs are in many ways reminiscent of signal flow graphs [8], although process graphs have no feedback and involve more complicated operations. A similar type of graph has previously been used to describe complicated weapon systems, e.g. [8]. In addition to their utility in determining data dependencies, a partial ordering of operations, and potential parallelism, process graphs can also be utilized to determine error bounds for a set of calculations. They are used primarily for this purpose, in a slightly different form, by McCracken and Dorn [9].

We propose that the type of flow diagram provided for a given algorithm should be an option of the specification writer. In some instances one of the two types, flow chart or process graph, may be more appropriate. In other instances it may be useful to include both a flow chart and a process graph. This leads to a possible

difficulty in that a flow chart yields a complete ordering on the operations performed, whereas a process chart ordinarily yields only a partial ordering. Some algorithms may be best described by a process graph involving computational blocks with decisions and iterations imbedded in them. These computational blocks might be best described by flow charts. Or the reverse may be better in some instances, although the flow of the data would seem to be somewhat obscured by the use of an overall flow chart.

2.2 Other

When the algorithm specification is being written it may be necessary to make certain assumptions about number representation or other details which may vary for some implementations, but have no profound effect on the algorithm. In this section such variations and their effect on the implementation should be pointed out.

2.3 References

This section contains a list of all books, reports, journal articles and other items which may be referenced for additional information.

3.0 Some Algorithms

The following examples are intended to illustrate the proposed method of algorithm specification. While not all types of algorithms are included, among them are purely a computational algorithm, a purely manipulative algorithm, and several floating point algorithms which involve a number of tests and shifts as well as a few arithmetic operations.

I. Coordinate Rotation: Airframe to Earth Coordinates

1.0 Background Information and General Discussion

The purpose of this algorithm is to convert the coordinates of a radar target or other object given in airframe coordinates to earth coordinates.

1.1 Other Algorithms Used

This algorithm uses the sine and cosine algorithms.

1.2 Input Values

The input values to this routine are the coordinates of the target with respect to the airframe, and the heading, pitch angle, and roll angle of the airframe. All input values are real numbers. It is assumed that the heading, pitch angle, and roll angle are given in degrees. The heading, ψ normally lies in the range $[0, 360^{\circ}]$, while the pitch angle, θ and the roll angle, ϕ lie in the range $[-180^{\circ}, 180]$. No error occurs for angles outside this range, A due north heading is zero, while easterly headings are positive. Pitch angles are positive nose up, and roll angles are positive right wing down.

1.3 Output Values and Accuracy

The output values are the earth coordinates of the target with respect to the position of the airframe. Output units are the same as the input units. The positive directions for the output coordinates are north, east, and down, respectively.

The accuracy of the results depens on the accuracy of the sine/cosine algorithm and could be as large as about 7 or 8 times the error in that algorithm, relative to the largest of the airframe coordinates. The usual error will be much smaller, perhaps one or two bits compared to the largest of the airframe coordinates.

1.4 Operation Count

The algorithm requires six sine/cosine evaluations, 26 multiplications, and 10 additions which should dominate the total execution time.

1.5 Memory Requirements

This routine requires about ten temporary storage locations, one constant, and locations for any registers that must be saved.

2.0 Mathematical and Natural Language Description

Let $x_a - y_a - z_a$ be a right hand orthogonal coordinate system associated with the airframe. x_a is positive forward along the airframe center line, y_a is positive out the right wing, and z_a is positive downward. Let x_a, y_a, and z_a represent the airframe coordinates of the target. Let ψ be the airframe heading with respect to true north, θ the airframe pitch angle--positive upward, and ϕ the airframe roll angle - positive right wing down. Then the target position in earth coordinates is given by a rotation of the position in airframe coordinates, represented by the matrix multiplication

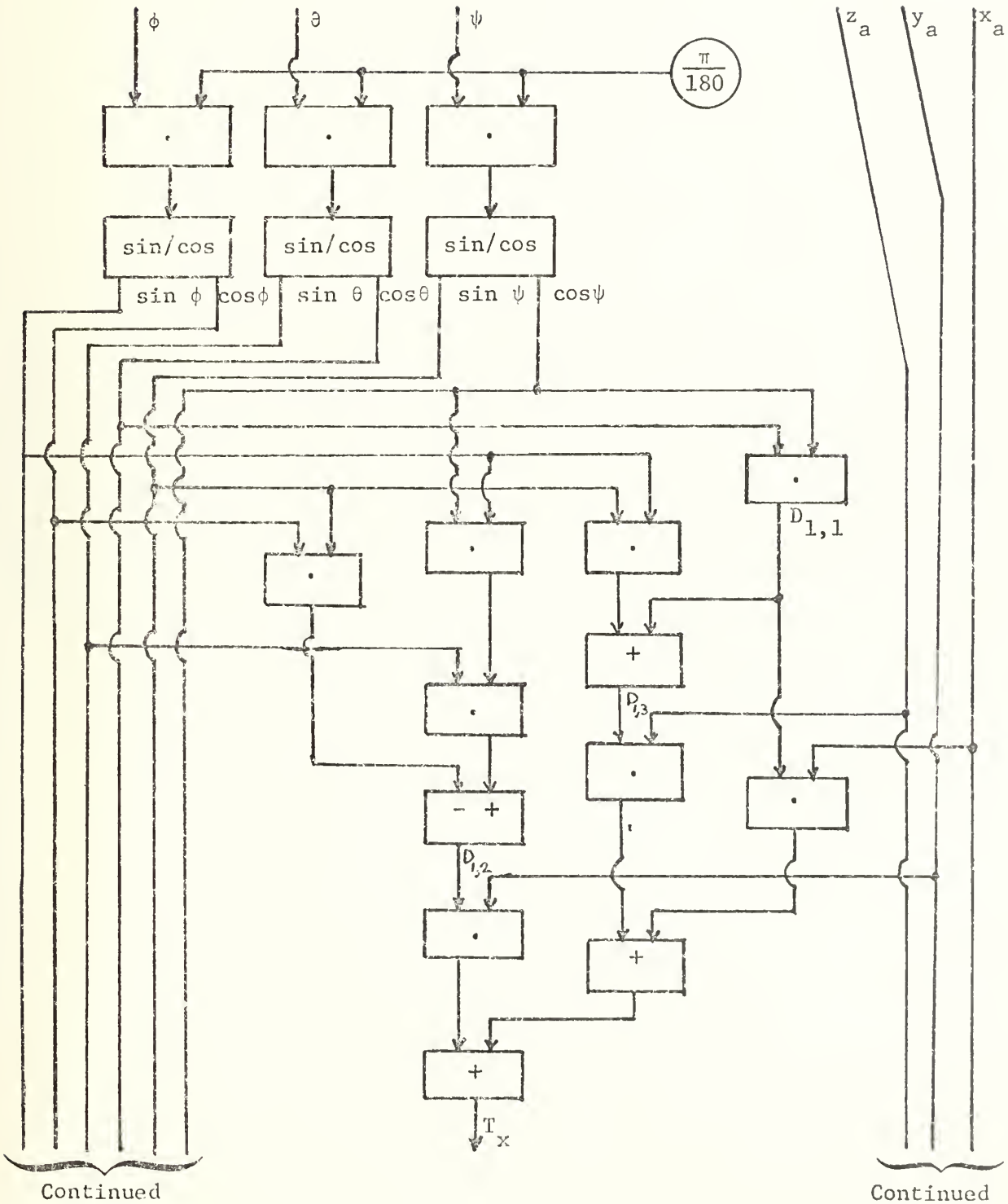
$$\begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} = \begin{pmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{pmatrix} \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix}$$

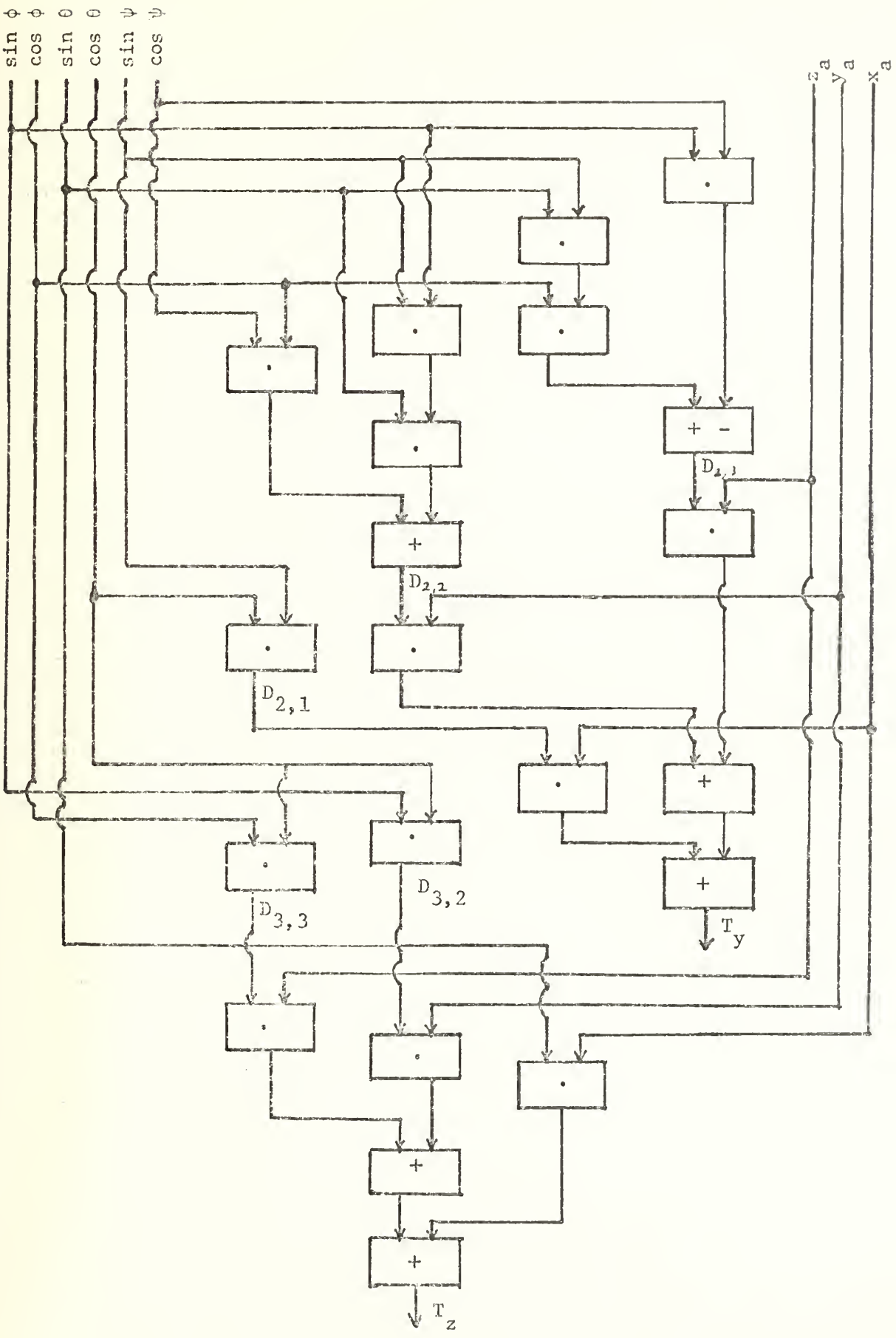
. Here T_x, T_y and T_z are the north, east, and down components of the earth coordinates, respectively. The matrix elements are given by

$$D_{11} = \cos\psi \cos\theta$$
$$D_{12} = \sin\phi \sin\theta \cos\psi - \sin\psi \cos\phi$$
$$D_{13} = \sin\phi \sin\psi + \cos\theta \cos\psi$$
$$D_{21} = \cos\theta \sin\psi$$
$$D_{22} = \cos\phi \cos\psi + \sin\phi \sin\theta \sin\psi$$
$$D_{23} = \cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi$$
$$D_{31} = \sin\theta$$
$$D_{32} = \sin\phi \cos\theta$$
$$D_{33} = \cos\phi \cos\theta$$

2.1 Flow Diagram

2.1.1 Process Graph




sin φ
cos φ
sin θ
cos θ
sin ψ
cos ψ
z_a
y_a
x_a
$D_{2,3}$
$D_{2,2}$
$D_{2,1}$
$D_{3,3}$
$D_{3,2}$
T_y
T_z

2.2 Other

None

2.3 References

See Bibliography [12]

II. Coordinate Conversion: Spherical to Cartesian Coordinates

1. Background Information and General Discussion

The purpose of this algorithm is to convert the given spherical coordinates of a point in three space to the corresponding cartesian coordinates.

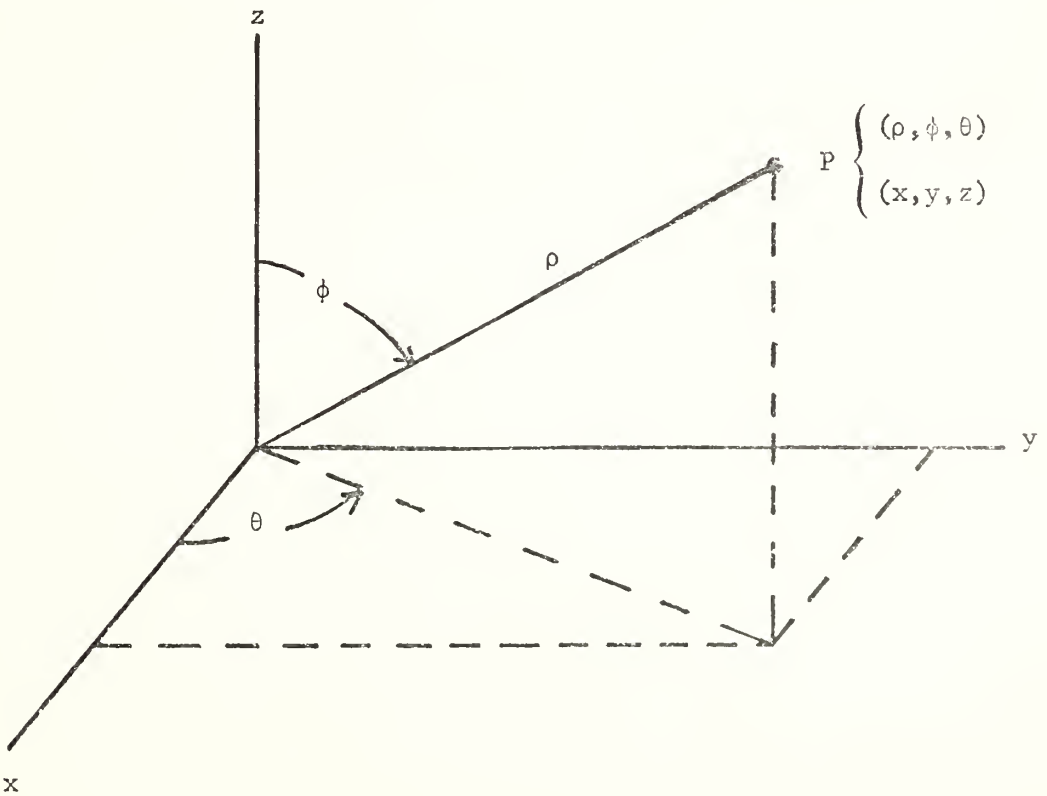
1.1 Other Algorithms Used

The sine/cosine algorithm is used by this algorithm.

1.2 Input Values

The input values are the spherical coordinates, (ρ, ϕ, θ) of a given point, which are real numbers. Here ρ is the distance of the point from the pole, or origin, ϕ is the angle the radius vector makes with the z-axis, and θ is the angle the projection of the radius vector into the x-y plane makes with the x-axis, measured positive counterclockwise. Angles are measured in radians.

The figure shows the relationship for a typical point P.



1.3 Output Values and Accuracy

The output values are the x, y, and z coordinates of the given point. The units are the same as those of the input ρ. The error in the coordinates, relative to ρ, would be no more than twice the error in the sine/cosine algorithm.

1.4 Operation Count

This algorithm requires four sine/cosine evaluations and four multiplications.

1.5 Memory Requirements

Approximately four locations are required for temporary storage plus any locations required for storage of registers.

2.0 Mathematical and Natural Language Description

The equations relating spherical coordinates (ρ, ϕ, θ) and cartesian coordinates (x, y, z) are

$$x = \rho \sin \phi \cos \theta$$

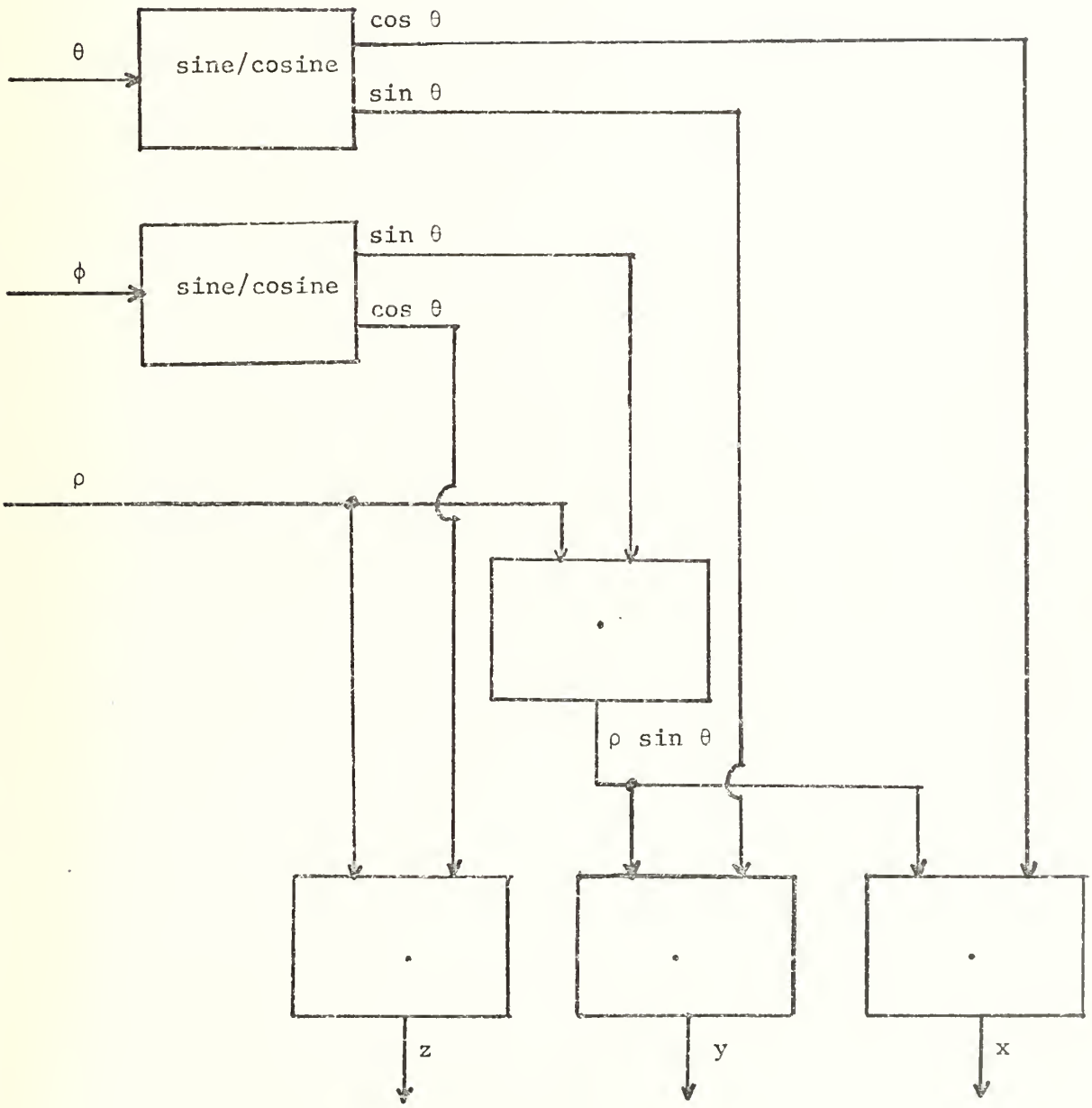
$$y = \rho \sin \phi \cos \phi$$

$$z = \rho \cos \phi .$$

ρ, ϕ, and θ are real numbers and ordinarily $\rho \geq 0$, $0 \leq \phi \leq \pi$, and $0 \leq \theta < 2\pi$, although these restrictions are not necessary for proper operation of the algorithm.

2.1 Flow Diagram

2.1.1 Process Graph



2.2 Other

None

2.3 References

None

III. Radar Target Processing

1.0 Background Information and General Description

The purpose of the algorithm is to convert the radar outputs of range, bearing, and elevation with respect to airframe coordinates to earth coordinates.

1.1 Other Algorithms Used

The spherical to cartesian coordinate conversion algorithm (II) and the coordinate rotation algorithm (I) are used.

1.2 Input Values

The input values to the algorithm are the radar outputs of range, bearing, and elevation, and the airframe heading, pitch angle, and roll angle. All angles are in degrees. Positive directions for the airframe are as described in algorithm I. Target bearing is measured from the airframe $x_a - z_a$ plane, positive clockwise when viewed from above. Elevation is positive upward with respect to the airframe $x_a - y_a$ plane.

1.3 Output Values

The output values are the earth coordinates of the target with respect to the airframe position, in northerly, easterly, and downward components. The units are the same as those of the input range. The error in the output values is no greater than the sum of the errors in the algorithms I and II, hence the error is no greater than 8 or 9 times the error in the sine/cosine routine. This is the error measured relative to the input value of range. In the usual case the error should be about one or two bits, relative to the range.

1.4 Operation Count

The algorithm requires two additions and two multiplications for argument processing before one transfer to each of the algorithms mentioned in section 1.1.

1.5 Memory Requirements

Two constants are required by the algorithm.

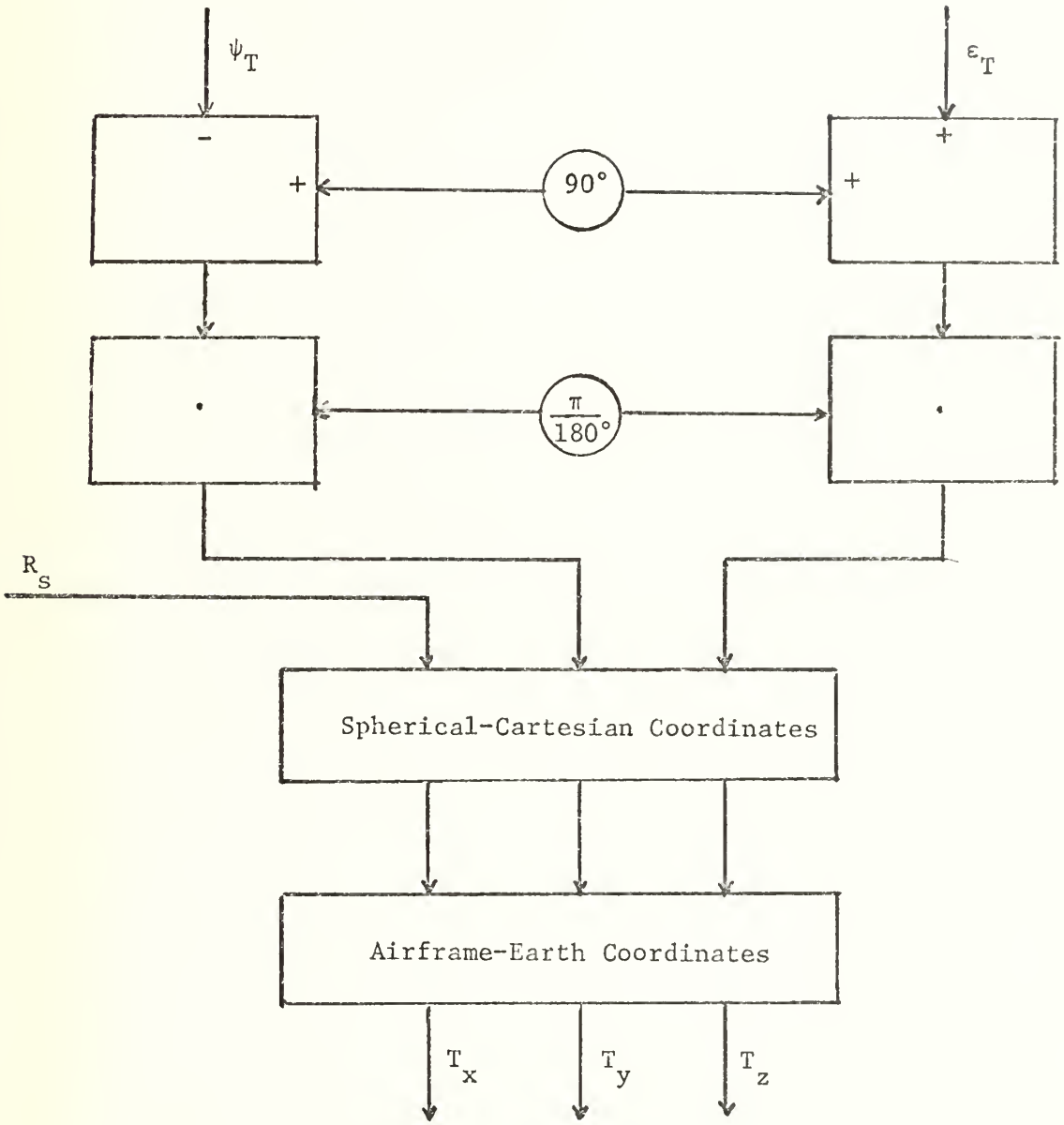
2.0 Mathematical and Natural Language Description

Let x_a, y_a, z_a, be the axes of an orthogonal coordinate system associated with the airframe. x_a is positive forward along the airframe center line, y_a is positive out the right wing and z_a is positive downward. Let R_s be the slant range to the target, ε_t the target elevation with respect to the $x_a - y_a$ plane-positive upward, and ψ_t the target bearing with respect to the $x_a - z_a$ plane-positive clockwise when viewed from above. Then the spherical coordinates of the target are (R_s, ϕ, θ), where $\phi = 90 + \varepsilon_t$ and $\theta = 90^{\circ} - \psi_t$.

The angles are assumed to be in degrees and are converted to radians before entry to the spherical to cartesian coordinate conversion algorithm. The airframe coordinates are then converted to earth coordinates using algorithm I.

2.1 Flow Diagram

2.1.1 Process Graph



2.2 Other

None

2.3 References

See Bibliography [12]

IV. Floating Point Multiplication

1.0 Background Information and General Description

The purpose of this algorithm is to obtain the product of two numbers represented in floating point format on a computer which does not have floating point hardware. It is assumed the computer is capable of performing fixed point additions and multiplications.

1.1 Other Algorithms Used

No other algorithms are used by the floating poing multiply.

1.2 Input Values

The input values are the two operands expressed in floating point format.

1.3 Output Values and Accuracy

The output value is the product of the operands expressed in floating point format. The output valve is in normalized form provided the input values are in normalized form (no leading zero digits). If the characteristic of the product exceeds the maximum allowable, an overflow is indicated and the characteristic is set to the maximum number which can be represented. The result is accurate to $\pm \frac{1}{2}$ in the least significant digit of the mantissa.

1.4 Operation Count

The number of operations depends on the instruction repertoire of the computer, but will involve one multiplication, one addition, and about 15 index incrementation, test, compare, shift, and logical operations.

1.5 Memory Requirements

The algorithm requires about ten locations for temporary storage and constants, depending on how the arguments are transmitted to the algorithm.

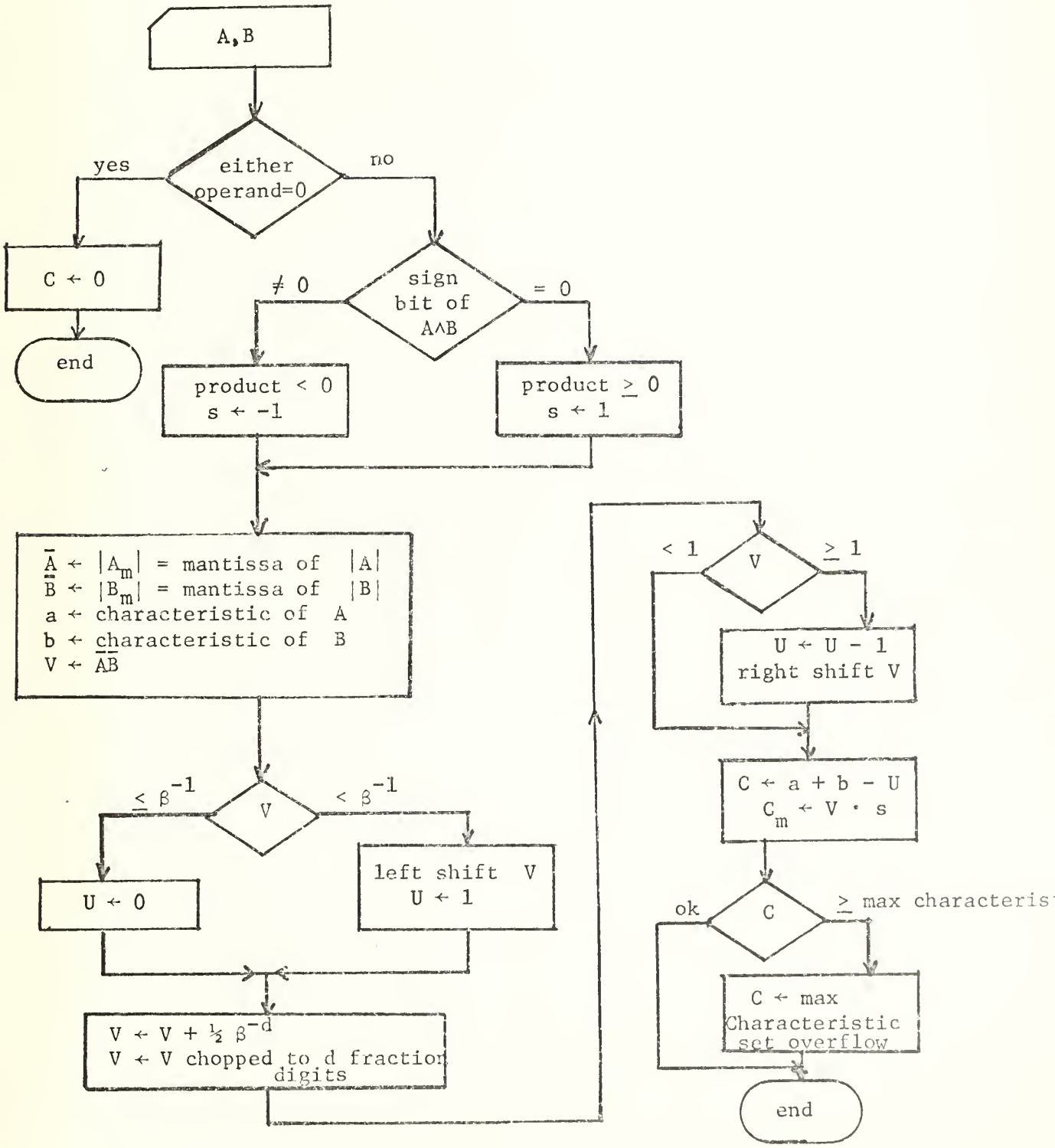
2.0 Mathematical and Natural Language Description

Let the two operands be represented by A and B . Define A_m and a such that $A = A_m \beta^a$, where β is the base of the number system used by the computer, A_m is the mantissa, and a is the characteristic. We assume A_m is in normalized form that is $\beta^{-1} \leq |A_m| < 1$ or $A_m = 0$. B_m and b are defined analogously. Because $AB = A_m B_m \beta^{a+b}$ the algorithm multiplies mantissae, adds characteristics, normalizes the product of the mantissae, rounds and renormalizes the product of the mantissae. Normalizations are done under the assumption A and B were normalized. If A or B are not normalized, then the product may not be normalized. If the product of the (unnormalized) mantissae has more than d + 1 (d = number of significant digits in the mantissa of the representation) leading zeros, the product will be zero. No error indication occurs in this instance.

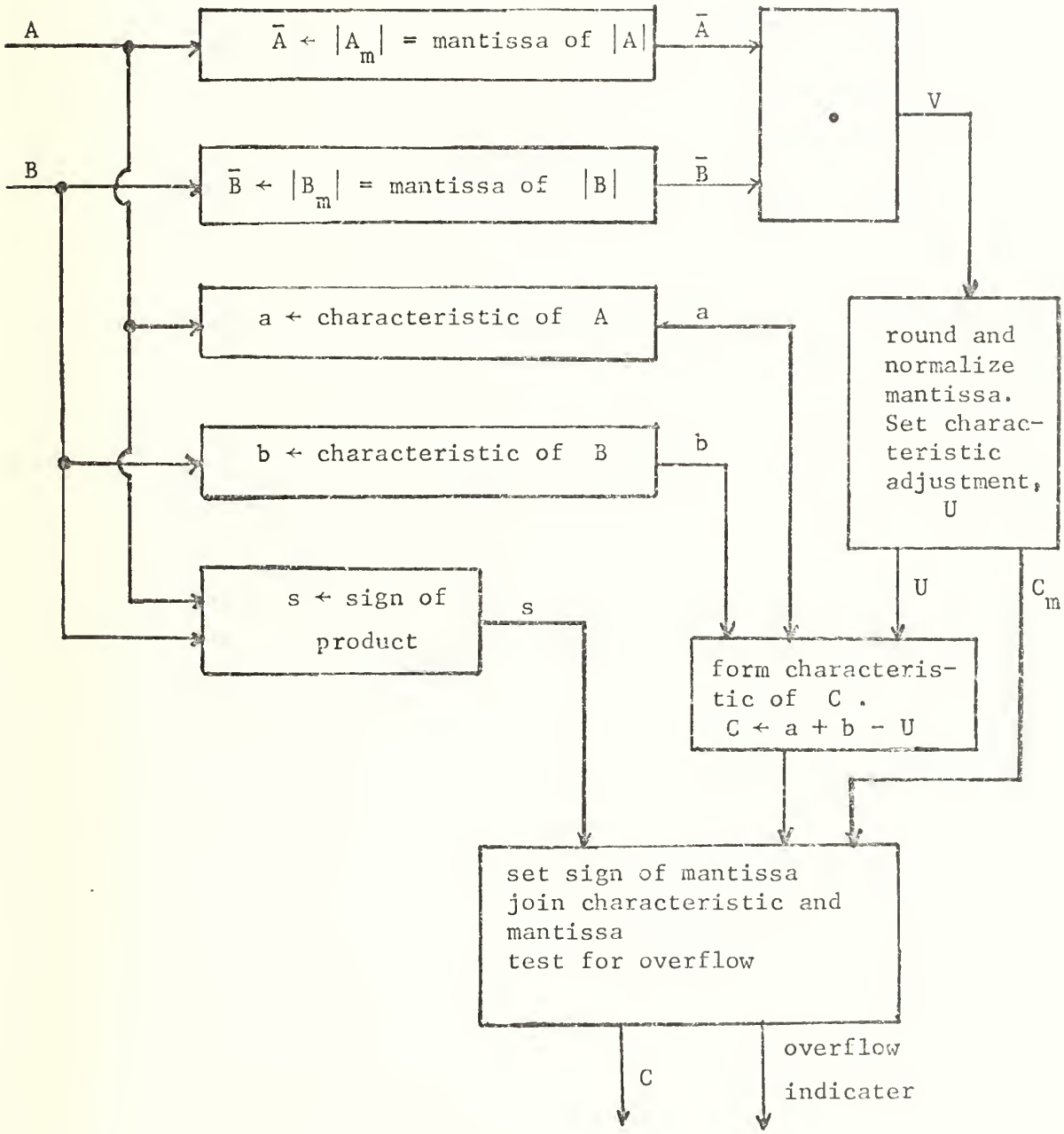
If overflow occurs the characteristic is set to the largest value possible and an overflow indicator is set.

2.1 Flow Diagram

2.1.1 Flow Chart



2.1.2 Process Graph



2.2 Other

We have assumed that the floating point representation is of a particular form, i.e., the mantissa normalized between β^{-1} and 1. Other normalizations are possible, in particular the mantissa might be normalized between 1 and β. In this instance the procedure for normalization might involve a right shift rather than a left shift.

Another slight difference which can occur is the bias of the characteristic. In order to avoid storing negative characteristics, the value stored is biased by a positive integer, α. If M represents the maximum value which can be represented by the digits allowed for the characteristic, the approximate range of numbers which can be represented is $\beta^{-\alpha}$ to $\beta^{M-\alpha}$. Hence, α is often chosen to be near M/2. In any case, if $\alpha \neq 0$ the bias must be allowed for when computing the characteristic of the product.

2.3 References

None

V. Floating Point Addition

1.0 Background Information and General Description

The purpose of this algorithm is to obtain the sum of numbers represented in floating point format on a computer which does not have floating point hardware. It is assumed the computer can perform fixed point additions.

1.1 Other Algorithms Used

None

1.2 Input Values

The input values are the two operands expressed in floating point format.

1.3 Output Values and Accuracy

The output value is the sum of the two input values, expressed in normalized floating point format. The output value is accurate to $\pm \frac{1}{2}$ in the d^{th} (least) significant digit provided the input values are in normalized form. Unnormalized input values may result in less accuracy. Underflows (number too small to be represented) result in the output value being set to zero, while overflows (number too large to be represented) result in the output value being set to the largest number which can be represented, with the appropriate sign.

1.4 Operation Count

The operation count depends on the instruction repertoire of the computer, but will involve between 3 and 3 + d additions, between 5 and 5 + d tests, and a similar number of shifts. All but two of the additions can be performed as index increments. At least five logical operations will be required.

1.5 Memory Requirements

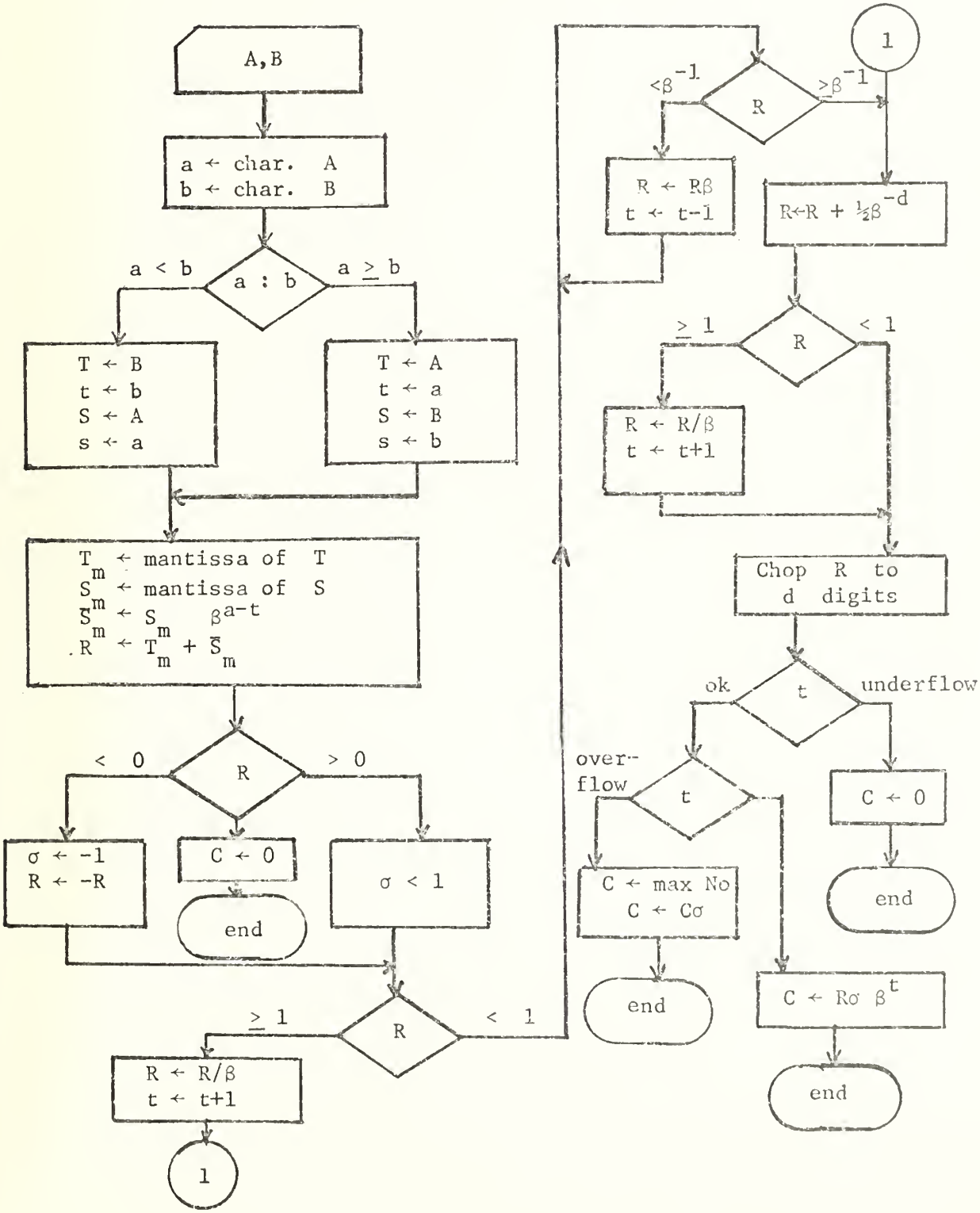
The algorithm requires about ten locations for temporary storage and constants.

2.0 Mathematical and Natural Language Description

Let A and B be represented in floating point form. Define A_m and a such that $A = A_m \beta^a$, where $\beta^{-1} \leq A_m < 1$ or $A_m = 0$. Here β is the base of the number system used by the computer. B_m and b are defined analogously. Let T be the number with larger exponent, or either one if they have the same exponent, and let S be the other. Define T_m, t, S_m and s as above. Then $s \leq t$ and the tentative, unrounded, mantissa of A + B is $R = T_m + S_m \beta^{s-t}$. If $R = 0$, $C = A + B = 0$. Otherwise $|R|$ is normalized, rounded to d digits, and renormalized if necessary. The characteristic of the sum is adjusted as $|R|$ is normalized, then tested for underflow or overflow. On underflow, the output value is set to zero, on overflow the output value is set to the largest number which can be represented. Finally the appropriate sign is attached to the (nonzero) result.

2.1 Flow Diagram

2.1.1 Flow Chart



2.2 Other

Floating point number representations different than that assumed are possible, e.g. the mantissa normalized to lie between 1 and β. In this instance the tests for normalization may be slightly different.

2.3 References

None

VI. Floating Point Compare

1.0 Background Information and General Description

The purpose of this algorithm is to compare two numbers expressed in floating point format and determine if they are equal, or which is larger if they are not equal.

1.1 Other Algorithms Used

None

1.2 Input Values

The input values are the two numbers A and B in normalized floating point form.

1.3 Output Values and Accuracy

The output value is an integer which takes on the following values:

$$r = \begin{cases} 0 & \text{if } A = B \\ 1 & \text{if } A > B \\ -1 & \text{if } A < B \end{cases}$$

1.4 Operation Count

The algorithm may include as many as 4 tests, 5 logical operations, and a number of load and store instructions.

1.5 Memory Requirements

Two constants are required and up to four temporary storage locations, depending of the number of registers and whether they must be saved.

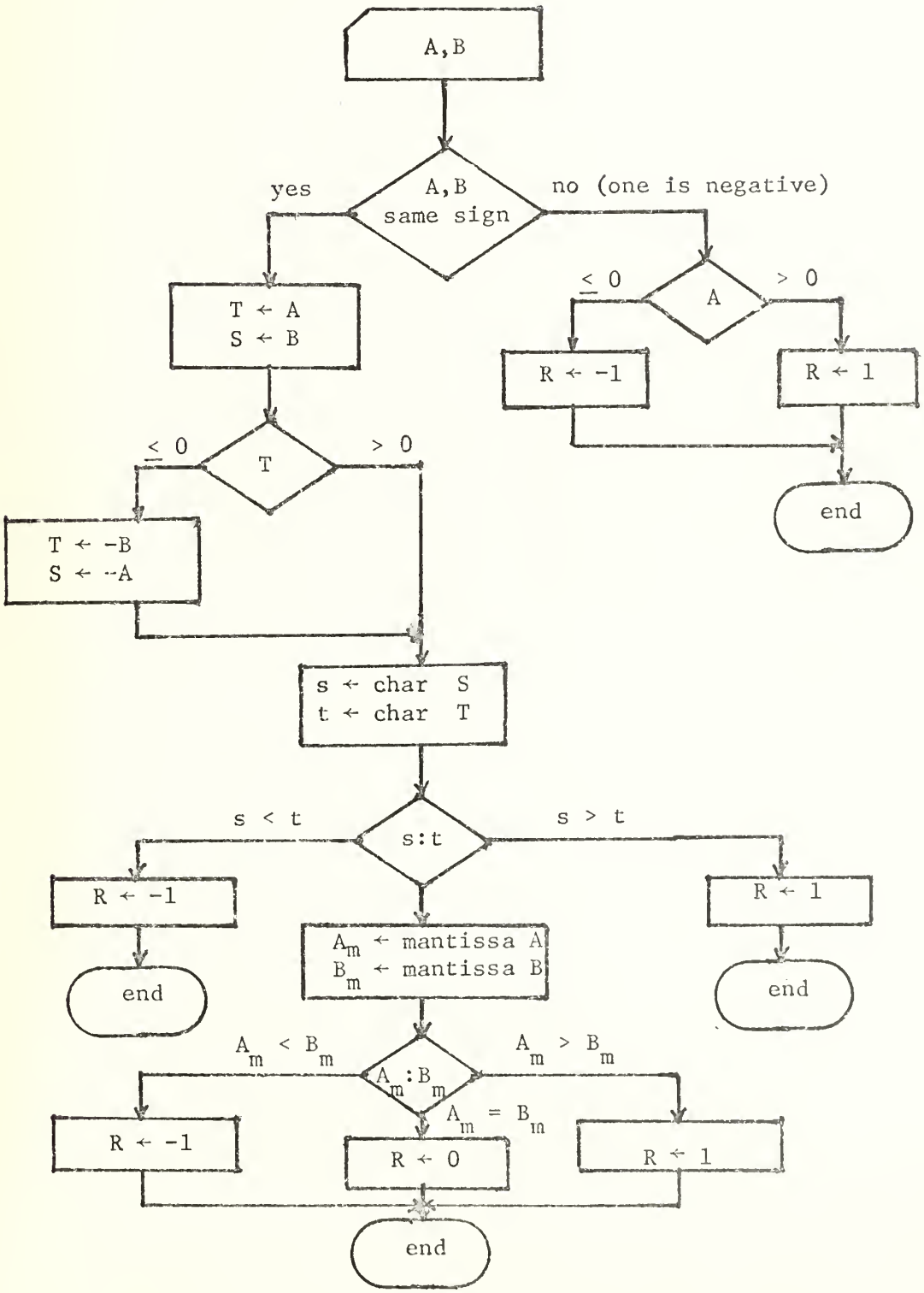
2.0 Mathematical and Natural Language Description

Let A and B be represented in normalized floating point form. Define $A = A_m \beta^a$ where $\beta^{-1} \leq A_m < 1$ or $A_m = 0$ and β is the base of the number system used by the computer. B_m and b are analogously defined. If A and B have opposite sign bits, then $r = 1$ or -1 as $A \geq 0$ or $A > 0$, respectively. When they have the same sign, the characteristics are compared and $r = \sigma$ or $-\sigma$ as $a > b$ or $a < b$,

where $\sigma = 1$ or -1 as A is nonnegative or negative, respectively. If $a = b$, the mantissae are compared and $r = 1, 0,$ or -1 as $A_m > B_m$, $A_m = B_m$, or $A_m < B_m$, respectively.

2.1 Flow Diagram

2.1.1 Flow Chart



2.2 Other

None

2.3 References

None

VII. Memory Rotate

1.0 Background Information and General Discussion

The purpose of this algorithm is to rotate the contents of a table in memory an arbitrary number of places.

1.1 Other Algorithms Used

None

1.2 Input Values

The input values are the tabular array, the number of entries in the table, n, and the number of places through which the table is to be rotated, r. It is assumed that $0 < r < n$. If r is outside this range an error may occur. No errors are indicated when this occurs, however.

1.3 Output Values and Accuracy

The output values are the rotated tabular array. There are no error indications.

1.4 Operation Count

There are principally three types of operations involved: (1) index incrementation, of which there are between $2n + 3$ and $3n + 3$; (2) tests, of which there are between $n + 3$ and $2n + 3$; and (3) load, store, and register exchange, of which there are about n each. The total number of operations will depend on the number of registers available.

1.5 Memory Requirements

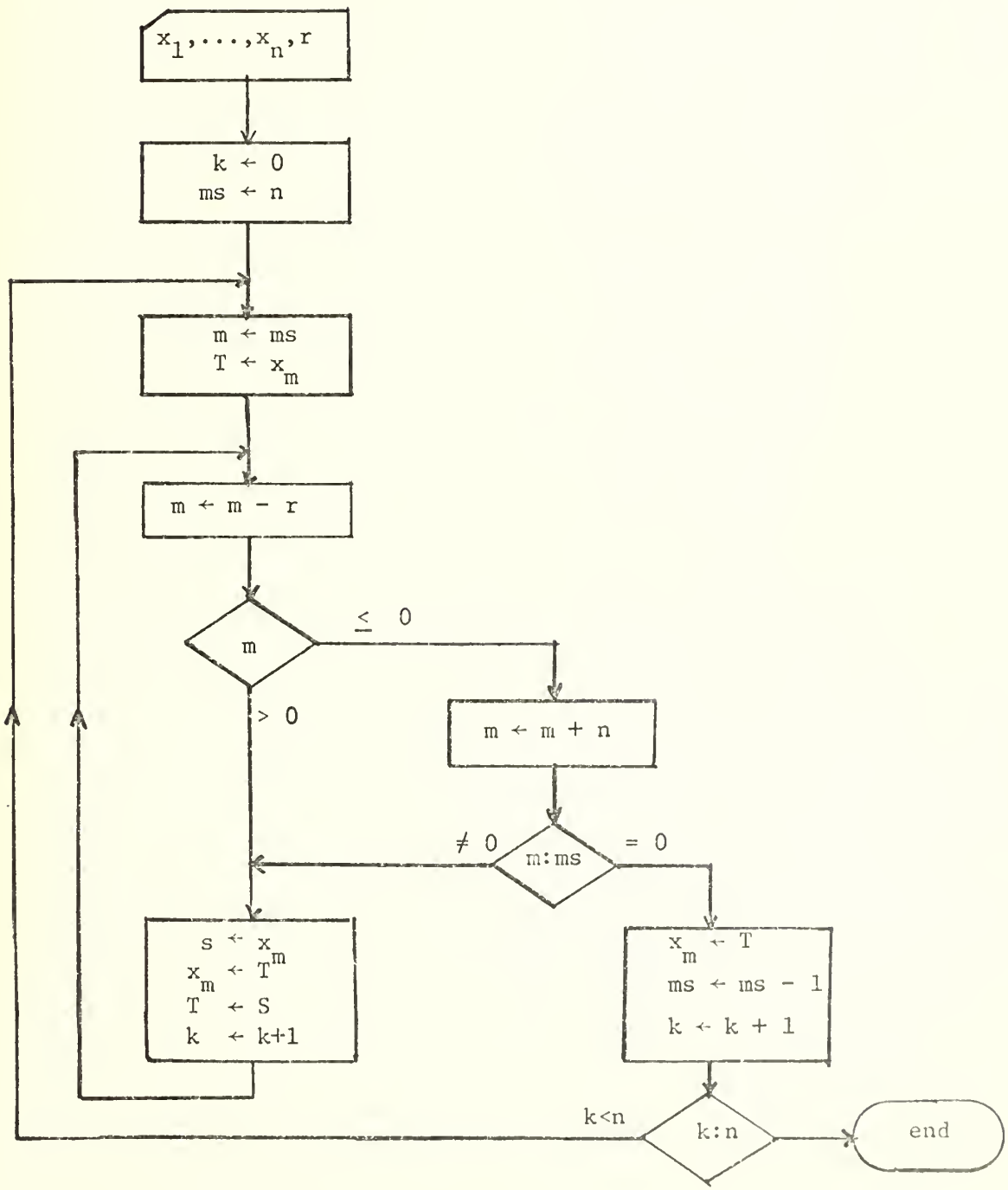
No more than five temporary storage locations should be required.

2.0 Mathematical and Natural Language Description

Let the table entries $x_0, \ldots, x_{n-1}$ be stored in memory locations $\ell, \ell + 1, \ell + 2, \ldots, \ell + n-1$. Then the algorithm performs the permutation resulting in $C(\ell + j) = x_i$, where $j = (i + r) \bmod n$, $i = 0, 2, \ldots, n-1$. Here $C(k)$ indicates the contents of location k.

2.1 Flow Diagram

2.1.1 Flow Chart



2.2 Other

None

2.3 References

None

4.0 Afterthoughts

Some difficulties remain which must ultimately be resolved, and there are related problems which have not been discussed.

In some instances the number system used by the computer may very drastically affect the algorithm for performing certain functions. At this point it is no longer possible to keep the algorithm independent of the machine architecture. Perhaps these very basic algorithms should be considered to be essentially a part of the computer and not documented in the same fashion as higher level algorithms. The floating point algorithms are very near to low level algorithms in that sense.

In a hierarchy of algorithms some uniform system of handling errors must be used, particularly in instances where there must be no crashes, that is, some reasonable recovery must be made from all errors. This subject has not been discussed previously, but it seems to the writer that error recovery must be handled at a point as near to the discovery of the error as possible. Related problems are multiple errors, and the problem of re-initialization if an error occurs part way through a computation. That is, if an algorithm updates variables, the old value must not be lost until it is certain the calculation can be completed.

The problem of how to specify recursive functions has not been explored, but must be considered. Likewise the problem of partitioning a large system into algorithms which will be useful to other systems (or already used by other systems) must be considered and solved.

The weakest link in any specification is likely to be the individuality exhibited by the person writing the specification. 'Favorite' algorithms are likely to be given much better treatment than algorithms which are understood poorly, or disliked by the author. In other instances, algorithms thoroughly understood by the specifier may be poorly written because they are so 'simple'.

Those tendencies can only be overcome by very stringent guidelines on the specification, certainly more stringent than those proposed in Section 2.1.

Bibliography

1. John Bruno and Kenneth Steiglitz, "The Expression of Algorithms by Charts," pp 97-115 in Algorithm Specification, edited by Randall Rustin, Prentice-Hall, 1972

2. Jack B. Dennis and John B. Fosseen, "Introduction to Data Flow Schemas," MIT Project MAC, Computation Structures Group Memo 81-1, September 1973

3. Jack B. Dennis and David P. Misunas, "A Preliminary Architecture for a Basic Data-Flow Processor," pp 126-132 in Proc. 2nd Annual Symposium on Computer Architecture, 1975

4. Malcolm C. Harrison, "Declarative and Extendable Languages," pp 39-60 in Algorithm Specification, edited by Randall Rustin, Prentice-Hall, 1972

5. Anatol W. Holt, et al, "Information System Theory Project," RAC-TR-68-305, (AD 676 972), Rome Air Development Center, September 1968

6. Uno R. Kodres and William L. McCracken, "Design Study of an Avionics Navigation Microcomputer," pp 99-105 in Proc. 2nd Annual Symposium on Computer Architecture, 1975

7. P. R. Kosinski, "A Data Flow Programming Language," Report RC 4264 IBM T. J. Watson Research Center, Yorktown Heights, NY, March 1973

8. S. J. Mason, "Feedback Theory-Some Properties of Signal-Flow Graphs," Proc. IRE 41(1953) 1144-1156

9. D. J. McCracken and W. S. Dorn, Numerical Methods and Fortran Programming, Wiley, 1964

10. Naval Air Systems Command 01-85 ADF-2-10.1, Integrated Weapons System Theory for A6E Aircraft, 1971

11. D. L. Parnas, "A Technique for Software Module Specification with Examples" CACM 15 (1972) 330-336

12. James T. Pepe and John R. Nestor "Language Bendmark Tests," Draft Report, Intermetrics, Inc., Cambridge MA 02138, August 1973

13. J.E. Rodriguez, A Graph Model for Parallel Computation," Report TR-64, Project MAC, MIT, Sept 1969

14. Jacob T. Schwartz, "Principles of Specification Language Design with Some Observations Concerning the Utility of Specification Languages," pp 1-37 in Algorithm Specification, edited by Randall Rustin, Prentice-Hall, 1972

15. Robert M. Shapiro, et al, "The Representation of Algorithms" RADC-TR-69-313, Vol II, (AD 697026), Rome Air Development Center, Sept 1969

16. H. R. Strong, "Flowchartable Recursive Specification" pp 81-96 in Algorithm Specification, edited by Randall Rustin, Prentice-Hall, 1972

DISTRIBUTION LIST

	No. of Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Library Naval Postgraduate School Monterey, California 93940	2
Dean of Research, Code 023 Naval Postgraduate School Monterey, California 93940	2
Professor Richard Franke Department of Mathematics Naval Postgraduate School Monterey, California 93940	5
Professor Craig Comstock Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
Dr. Richard Lau Office of Naval Research Pasadena, California 91100	1
Professor Ladis D. Kovach Chairman, Department of Mathematics Naval Postgraduate School Monterey, California	1
Naval Electronics Laboratory Center 271 Catalina Blvd.	
Attn: Code 220	1
Code 5000	5
Mr. William J. Dejka	1
Mr. Norman Tinklepaugh	1
Mr. Don Eddington	1
Mr. David Harrison	1
Mr. Dwight Wilcox	1
San Diego, California 92152	